Prime Factorizations of Abstract Domains Using First-Order Logic

Elena Marchiori

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
and
University of Leiden, P.O. Box 9512, 2300 RA Leiden, The Netherlands
e-mail: elena@cwi.nl

Abstract. A methodology is introduced based on first-order logic, for the design and decomposition of abstract domains for abstract interpretation. First, an assertion language is chosen that describes the properties of interest. Next, abstract domains are defined to be suitably chosen sets of assertions. Finally, computer representations of abstract domains are defined in the expected way, as isomorphic copies of their specification in the assertion language. In order to decompose abstract domains, the notion of prime (conjunctive) factorization of sets of assertions is introduced. We illustrate this approach by considering typical abstract domains for ground-dependency and aliasing analysis in logic programming.

1 Introduction

In the theory of abstract interpretation [3], abstract domains are (computer) representations of properties. The semantics of an abstract domain is given by a function called concretization, that maps elements of the abstract domain into elements of a 'concrete domain'. Two fundamental aspects of the study of abstract domains are the investigation of representations supporting efficient implementations, and the comparative analysis of the properties represented by abstract domains. This paper is concerned with the latter aspect.

Previous work on this subject is mainly based on two equivalent techniques (cf. [3]): Galois connections and closure operators. In [3] comparison of abstract domains is defined by means of the notion of abstraction, where an abstract domain is more abstract than another one if there is a Galois insertion from the first into the latter. This notion is weakened in [7], where the comparison is defined w.r.t. a given property, by means of the notion of quotient of one abstract domain w.r.t. another one, describing the part of the former abstract domain that is useful for computing the information described by the latter one. In [4], the approach based on closure operators is used for investigating domain complementation in abstract interpretation. The authors formalize the concept of decomposition of an abstract domain, as a set of abstract domains whose reduced product yields the initial abstract domain and use the notion of pseudo-complement for decomposing abstract domains.

In this paper we propose a method based on first-order logic for the design and decomposition of abstract domains. First, an assertion language is chosen whose syntax specifies the properties of interest, and whose semantics is fixed by means of a structure characterizing the meaning of the predicates in accordance with the properties they are supposed to describe. Next, an abstract domain is defined to be a suitably chosen set of assertions. Finally, computer representations of abstract domains are defined in the expected way, i.e., they have to respect (i.e., be isomorphic to) their specification in the assertion language. In order to decompose abstract domains, the notion of prime (conjunctive) factorization of sets of assertions is introduced. This is a standard algebraic notion of factorization, where an abstract domain is factorized in pairwise 'disjoint' parts.

This method has various benefits. First, it allows one to focus only on the abstract domains that describe the properties of interest, that are those expressible in the chosen assertion language. This is not the case for the standard methods above mentioned, where all possible abstract domains (on the concrete domain) are taken into account. Moreover, using our method abstract domains can be decomposed in 'disjoint' factors. This desirable property is not guaranteed in the decompositions obtained using the approach of [4]. Finally, the two phases of design and computer representation of an abstract domain are neatly separated, where the design phase is performed at the logical level.

We illustrate this approach by considering typical abstract domains for ground-dependency and aliasing analysis in logic programming. The fragment $\mathcal{L}$ of a first-order assertion language introduced in [13] (actually, a slight extension of this) is used. Logical descriptions of various abstract domains are given: *Def* [8] and *Pos* [14, 15] for ground-dependency analysis; *Sharing* [10] and *ASub* [18] for aliasing analysis. Maximal factorizations for these domains are obtained by inspecting the structure of the assertions in the abstract domains, and they are used for analyzing and comparing the abstract domains.

The paper is organized as follows. The next section introduces our methodology. Section 3 presents an assertion language for the design of typical abstract domains for logic programming, and Section 4 contains a comparative study of various abstract domains for logic programming. Finally, Section 5 contains a discussion on related work and some conclusive remarks.

2 Abstract Domains in Assertion Form

First-order logic is a familiar formalism, used for specifying as well as for reasoning about properties. We show in this section how first-order logic can be used for the design and decomposition of abstract domains for abstract interpretation.

Here and in the sequel $\mathcal{L}$ denotes a generic assertion language. We assume that the semantics of the predicates in $\mathcal{L}$ is fixed according to their intended meaning, by a given structure denoted by $\mathcal{M}$. Assertions are indicated by ϕ, ψ. As already mentioned, abstract domains represent properties of some syntactic objects, usually a subset of the variables of the considered program. Thus, the definition of abstract domain we give is parametric with respect to a set V

of syntactic objects. We adopt the following convenient assumptions: 1. V is (identified with) a set of distinct variables of $\mathcal{L}$; 2. in the definition of abstract domain, only the set of assertions of $\mathcal{L}$ whose free variables are contained in V is considered, denoted by $A(\mathcal{L}, V)$; 3. assertions with the same meaning are identified.

This last assumption amounts to consider equivalence classes of assertions of $A(\mathcal{L}, V)$, where $[\phi]$ denotes all the assertions that are logically equivalent to ϕ. For simplicity, in the sequel the squares in $[\phi]$ are often omitted.

Definition 1. (Abstract Domain on $\mathcal{L}$) An abstract domain (on $\mathcal{L}$), denoted by $\mathcal{A}$ (possibly subscripted), is a set of assertions of $A(\mathcal{L}, V)$ containing *false*, and closed under conjunction and variance[1]. □

Observe that this definition of abstract domain is consistent with the original one given by the Cousots (cf. [3]): the 'concrete domain' is the set of sets of valuations, and the 'concretization function' maps an assertion ϕ into the set of valuations that satisfy ϕ.

In the sequel, for simplicity, we shall often avoid to mention the element *false* when specifying the set of assertions of an abstract domain.

Example 1. A simple abstract domain for the study of the sign of program variables assuming integer values is given in [3]. For a considered set V of program variables, this domain can be specified in our formalism as follows: $\mathcal{L}$ contains the constants and function symbols of the program, and the unary predicates $\geq$, $\leq$; $\mathcal{M}$ maps terms into integers according with their intended interpretation, and specify the semantics of $\geq$, $\leq$ in the expected way. Then the abstract domain for the study of the sign of the variables in V can be described by the set $Sign_V$ of assertions that are conjunctions of atoms of the form $x \geq 0$, or $x \leq 0$, with x in V. □

Viewing abstract domains as sets of assertions allows us to use the following notion of (conjunctive) factorization for decomposing (in $\mathcal{L}$) an abstract domain in 'disjoint' parts.

The notation $\mathcal{A}_1 = \mathcal{A}_2$ is used, meaning that $\mathcal{A}_1$ and $\mathcal{A}_2$ contain the same equivalence classes. Moreover, for two sets $\mathcal{A}_1$, $\mathcal{A}_2$ of assertions, $\mathcal{A}_1 \wedge \mathcal{A}_2$ denotes the set $\{[\phi_1 \wedge \phi_2] \mid \phi_1 \in \mathcal{A}_1, \phi_2 \in \mathcal{A}_2\}$.

Definition 2. (Prime Factorization on $\mathcal{L}$) The set $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is a *(conjunctive) prime factorization* of $\mathcal{A}$ if the following conditions hold:
(a) If $n > 1$ then $\mathcal{A}_i \neq \{true, false\}$, for $i \in [1, n]$;
(b) for every $i \neq j$ $\mathcal{A}_i \cap \mathcal{A}_j = \{true, false\}$;
(c) $\mathcal{A}_1 \wedge \ldots \wedge \mathcal{A}_n = \mathcal{A}$.

We call $\mathcal{A}$ *reduced* if it has only one factorization. Moreover, a factorization of $\mathcal{A}$ is *maximal* if $\mathcal{A}_i$ is reduced, for $i \in [1, n]$. □

[1] Recall that a variant of an assertion ϕ is any assertion $\phi\sigma$ obtained by applying to ϕ a substitution σ that renames the variables of ϕ

The name 'prime' in the above definition is used to underline the similarity of our definition with the standard algebraic notion of factorization of integers in relatively prime factors. For simplicity, in the sequel we write 'factorization' instead of 'prime factorization'. Clearly, if $\mathcal{A}$ is reduced then $\{\mathcal{A}\}$ is its only factorization, and it is maximal.

Example 2. It is easy to check that $\{Sign_{\leq 0}, Sign_{\geq 0}\}$ is a maximal factorization of $Sign_V$, where $Sign_{\leq 0}$ is the set of assertions that are conjunctions of atoms of the form $x \leq 0$, with x in V, and where $Sign_{\geq 0}$ is defined analogously. □

In order to improve the precision of the static analysis of logic programs, abstract domains can be composed by means of the notion of reduced-product ([3]). Intuitively, the reduced product of two domains is obtained from the cardinal product of the domains by identifying pairs of elements whose conjunction represent the same information. A factorization yields a reduced-product in the expected way.

Proposition 3. *If $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ is a factorization of $\mathcal{A}$ then $\mathcal{A}$ is the reduced-product of $\mathcal{A}_1, \ldots, \mathcal{A}_n$.*

Proof. Let *Val* denote the set of valuations. $\mathcal{M}$ provides a Galois insertion of an abstract domain $\mathcal{A}$ into the concrete domain 2^{Val} consisting of sets of valuations. This Galois insertion is determined by the concretization function $\gamma_{\mathcal{A}}$ that maps an assertion ϕ of $\mathcal{A}$ into the set of valuations that satisfy ϕ. Observe that $\gamma_{\mathcal{A}}$ is injective because equivalent assertions are identified. Then the operator $\wedge$ on abstract domains (on $\mathcal{L}$) is a reduced-product operator. □

The benefit of using this first-order framework is that the definition, decomposition and comparison of abstract domains can be performed in a uniform and familiar setting. However, (computer) representations of abstract domains for their efficient manipulation ([9]) often need different lattice structures (see, e.g., [2] for ground-dependency analysis). Therefore the notion of representation of an abstract domain is defined as follows. First, we need some preliminary terminology. The following notion of embedding of an abstract domain into $\mathcal{L}$ is used. Here and in the the sequel $\mathcal{D}$ denotes an abstract domain (on any complete lattice) and $\gamma_{\mathcal{D}}$ denotes its concretization function (cf. [3]).

Definition 4. (Embedding) An *embedding* of $\mathcal{D}$ in $\mathcal{L}$ is an injective mapping $\varepsilon_{\mathcal{D}} : \mathcal{D} \to \mathcal{L}$ s.t. for every D in $\mathcal{D}$, α is in $\gamma_{\mathcal{D}}(D)$ if and only if $\varepsilon(D)$ is true under α. □

Thus an embedding of a domain into $\mathcal{L}$ consists of the (equivalence classes of the) assertions ϕ_D characterizing the sets $\gamma_{\mathcal{D}}(D)$ of valuations, with D in $\mathcal{D}$. The following result is an easy consequence of the definition of concretization function ([3]).

Proposition 5. *The image $\varepsilon_{\mathcal{D}}(\mathcal{D})$ of an embedding is an abstract domain on $\mathcal{L}$.*

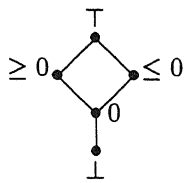
We can now formalize the concept of representation domain.

Definition 6. (Representation Domain) $\mathcal{D}$ is a *representation* of $\mathcal{A}$ (or equivalently $\mathcal{A}$ and $\mathcal{D}$ are *isomorphic*, denoted by $\mathcal{A} \simeq \mathcal{D}$) if there exists an *embedding* $\varepsilon_{\mathcal{D}}$ s.t. $\mathcal{A} = \varepsilon_{\mathcal{D}}(\mathcal{D})$.

□

The definition of representation domain clarifies the role of domains in assertion form, as those used in the design phase, in contrast to the representation domains used in the (efficient) implementation.

Example 3. Suppose $V = \{x\}$. Then a representation of $Sign_V$ is the familiar lattice pictured below



□

We conclude this section with a discussion on the relationship of our approach with the one based on closure operators. Following [3], one can associate with each abstract domain an (upper) closure operator (on sets of valuations) by means of the concretization function γ mapping an assertion into the set of valuations that satisfy that assertion. The closure operator associated with an abstract domain is the set of sets of valuations obtained by applying γ to each of its assertions. In the standard approach, also the vice versa holds, i.e., the lattice of abstract domains is isomorphic to the lattice of upper closure operators. This result does not hold when Definition 1 is considered, for two reasons. One is the hypothesis of closure under variance w.r.t. V: a set of valuations that is not closed under variance (w.r.t. V)[2] is an (upper) closure operator, but it is not an abstract domain (on $\mathcal{L}$). The other reason is related to the expressivity of the chosen assertion language $\mathcal{L}$: (the image of) a closure operator is an abstract domain (on $\mathcal{L}$) only if it can be described by means of a set of assertions (of $\mathcal{L}$). However, if one assumes that the assertion language allows to describe all sets of valuations closed under variance (w.r.t. V), then the lattice of abstract domains (according with Definition 1) is isomorphic to the lattice of upper closure operators on sets of valuations closed under variance (w.r.t. V).

[2] The notion of variant w.r.t. V of a set d of valuations is defined in the expected way: let ρ be a substitution that renames the variables of V with other variables of V. Then a variance of d is obtained by applying ρ to the domain of every valuation

3 Abstract Domains for Logic Programming

In this section, we show how a slight extension of the first-order assertion language $\mathcal{L}$ introduced in [13] can be used for the design and decomposition of typical abstract domains for the static analysis of logic programs.

Term properties, like groundness and sharing, have been identified as crucial when analyzing the run-time behaviour of logic programs. For instance, ground-dependency analysis can be used for compile optimization, by using matching instead of unification when it is known that at a given program point a variable is bound to a ground term every time the execution reaches that point. Information on the sharing among variables in a logic program is useful for important optimizations, like and-parallelism. The assertion language here considered allows to express properties of terms, like groundness, freeness, linearity, sharing, covering and independency. Informally, a term is *ground* if it does not contain variables, it is *free* if it is a variable, and it is *linear* if every variable occurs in it at most once. Moreover, a set of terms *share* if they have at least one common variable, while they are *independent* if they do not share. Finally, a term is *covered* by a set of terms if the set of its variables is contained in the union of the sets of variables of the terms in that set. For instance, the term $f(x,y)$ is covered by the set $\{g(x), g(y)\}$.

In order to define $\mathcal{L}$, a countable set *Var* of *(logical) variables* is used, denoted by v, x, y, z, possibly subscripted. Here and in the sequel, S represents a finite set of logical variables, and $|S|$ its cardinality. Moreover, the notation $S \subset S'$ indicates that S is a proper subset of S'.

Definition 7. (The Assertion Language) Let $\mathcal{L}'$ be the smallest set F of formulas containing atoms of the form $var(x)$, $ground(x)$, $linear(x)$, $share(S)$, and s.t. if ϕ_1 and ϕ_2 are in F then $\neg\phi_1$ and $\phi_1 \wedge \phi_2$ are also in F. The assertion language $\mathcal{L}$ consists of all the formulas of the form $\forall x_1, \ldots, x_n(\phi)$, with $\phi \in \mathcal{L}'$, and $n \geq 0$. □

The formula $\phi \vee \psi$ is used as a shorthand for $\neg(\neg\phi \wedge \neg\psi)$, $\phi \Rightarrow \psi$ denotes $\neg\phi \vee \psi$, and $\phi \Leftrightarrow \psi$ stands for $(\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi)$. Moreover, the propositional constants *true* and *false* are assumed to be in $\mathcal{L}$, where *true* is identified with the conjunction over the emtpy set of assertions $\wedge\emptyset$ and *false* with $\vee\emptyset$. In the sequel, the notation $share(x,y)$ is used as shorthand of $share(\{x,y\})$, with x, y distinct.

Observe that only a weak form of universal quantification is allowed, where $\forall$ does not occur in the scope of any $\neg$. For instance, $\forall z\,(var(z) \wedge \neg share(\{z,x\}))$ is in $\mathcal{L}$, but $\neg\forall z\,(var(z) \wedge \neg share(\{z,x\}))$ is not in $\mathcal{L}$.

The meaning of assertions in $\mathcal{L}$ is specified by means of the following structure $\mathcal{M}$. Let *OVar* be the set of *(object) variables*, here identified for simplicity with *Var*, and let *Fun* be a set of *functors with rank* (constants are identified with functors of rank 0). In the following, $occ(x,\tau)$ denotes the number of occurrences of the variable x in the term τ, and $OVar(\tau)$ the set of (object) variables occurring in τ.

Definition 8. The structure $\mathcal{M}$ contains the universe $\mathcal{U}$ consisting of the *(object) terms* built on $OVar$ and Fun. Moreover, for each predicate symbol p of $\mathcal{L}$, $\mathcal{M}$ contains a predicate in $\mathcal{U}$, also denoted by p, with the following semantics:

$$\begin{array}{ll}
\mathcal{M} \models var(\tau) & \text{if } \tau \in OVar \\
\mathcal{M} \models ground(\tau) & \text{if } OVar(\tau) = \emptyset \\
\mathcal{M} \models linear(\tau) & \text{if } occ(x, \tau) = 1 \text{ for every } x \text{ in } OVar(\tau) \\
\mathcal{M} \models share(\{\tau_1, \ldots, \tau_n\}) & \text{if } \bigcap_{i=1}^{n} OVar(\tau_i) \neq \emptyset
\end{array}$$

□

Example 4. The assertion $\neg share(\{x, y, z\}) \vee share(\{x, y\})$ is valid in $\mathcal{M}$. In fact, for every valuation α, if $OVar(x\alpha) \cap OVar(y\alpha) \neq \emptyset$ then $\mathcal{M} \models share(\{x, y\})\alpha$, otherwise $\mathcal{M} \models \neg share(\{x, y, z\})\alpha$. □

Note that even if *share* is not first-order (its argument is a set), it can be expressed in first-order logic by means of a family of first-order predicates $share_n$ of rank n, with $n \geq 0$. The set of valid (in $\mathcal{M}$) assertions of $\mathcal{L}$ has been characterized by means of a complete and decidable theory $\mathcal{T}$, by means of a simple axiomatization (see [13]).

The completeness and decidability of $\mathcal{T}$ provides an automatic tool for proving properties of some elements of an abstract domain, in the following way. In order to prove that an element ϕ of a domain satisfies a property P, specified in $\mathcal{L}$ by means of the assertion ψ, it is sufficient to check the validity of the implication $\phi \Rightarrow \psi$.

In order to use $\mathcal{L}$ for the static analysis of logic programs, it is necessary to assume that $\mathcal{U}$ contains the constants and function symbols of the considered class of programs. Moreover, we adopt the notation of the previous section: V denotes the set of (logical) variables representing the considered (program) variables, and $A(\mathcal{L}, V)$ the set of assertions of $\mathcal{L}$ whose free variables are contained in V. Therefore substitutions are identified with valuations.

An abstract domain (on $\mathcal{L}$) is specified according with Definition 1. Observe that we obtain a more specific notion of abstract domain than the original one (cf. [3]), because of the choice of the assertion language, and because of the condition of closure under variance. For instance, $\{ground(x), true, false\}$ would represent an abstract domain in the original definition, but it is not a legal one in our definition (unless $V = \{x\}$). The condition of closure under variance w.r.t. V has been implicitly assumed in the literature on abstract interpretation of logic programs, but it has not been taken into account when reasoning about these domains using the standard techniques based on Galois insertions or closure operator (cf. [4]).

We conclude this section with a simple example.

Example 5. Consider the abstract domain *Con* introduced by Mellish [17] and used in early mode and groundness analyzers [12]. *Con* consists of the bottom element $\bot$, and of the sets $S = \{x_1, \ldots, x_n\}$ of variables of V, with concretization function mapping $\bot$ into $\emptyset$ and $\gamma_{Con}(S) = \{\sigma \mid OVar(x\sigma) = \emptyset \text{ for all } x \in S\}$.

Let $\mathcal{A}_{Con}$ be the set of assertions that are conjunctions of atoms of the form $ground(x)$, with x in V. It is easy to show that $\mathcal{A}_{Con}$ satisfies Definition 1, and that Con is a representation of $\mathcal{A}_{Con}$, by considering the embedding ϵ_{Con} that maps $\bot$ into *false* and a set $\{x_1, \ldots, x_n\}$ into the assertion $ground(x_1) \wedge \ldots \wedge ground(x_n)$. □

4 Ground-Dependency and Aliasing Analysis

This section contains a comparative analysis of various abstract domains for the static analysis of logic programs, namely *Def*, *Pos*, *Sharing* and *ASub*. Each of these domains is shown to be the representation of an abstract domain on $\mathcal{L}$. These logical characterizations in $\mathcal{L}$ of the domains are used for deriving their maximal factorizations, for studying and comparing the original domains, as well as for defining new ones.

4.1 *Def* in Logical Form

The abstract domain *Def* was introduced by Marriott and Søndergaard for ground-dependency analysis in [15], based on previous work by Dart ([8]) on groundness analysis in deductive databases. We show that *Def* can be factorized into two reduced domains, describing groundness and covering, respectively.

First, we recall the definition of *Def*. *Def* is the largest class of positive boolean functions whose models are closed under intersection, augmented with the bottom element *false*. Recall that a boolean function F is positive if $F(true, \ldots, true) = true$. Here boolean functions are represented by (equivalence classes of) propositional formulas, as e.g. in [15]. In order to define the concretization function γ_{Def}, substitutions are viewed as truth assignments as follows. For a substitution σ, the truth assignment $groundsσ$ maps a propositional variable x to *true* iff $x\sigma$ is ground, and to *false* otherwise. Moreover, the notion of instance σ' of a substitution σ is used, meaning that σ' is obtained by composing σ with some substitution. The concretization function γ_{Def} maps an element F of *Def* into the set $\gamma_{Def}(F)$ of those substitutions σ s.t. for every instance σ' of σ, F under the truth assignment $groundsσ'$ is *true*. Intuitively, $\gamma_{Def}(F)$ extracts the 'monotonic' (in the sense that its truth is preserved under instantiation) information described by the propositional formula F.

Consider the following abstract domain $\mathcal{A}_{Def}$ on $\mathcal{L}$.

Definition 9. $\mathcal{A}_{Def}$ is the set of assertions that are conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z,x) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_n))$, with $n \geq 0$, where $x, y_1, \ldots, y_n$ are in V, and z is a fresh variable. □

We show that *Def* is a representation of $\mathcal{A}_{Def}$, and provide a maximal factorization of $\mathcal{A}_{Def}$.

First, *Def* is characterized in logical form by means of the following transformation. We use the representation of an element F in *Def* as a conjunction of formulas, called definite clauses, of the form $y_1 \wedge \ldots \wedge y_n \rightarrow x$ with $n \geq 0$ (see [8, 2]).

Definition 10. The transformation $\varepsilon_{Def} : Def \rightarrow \mathcal{L}$ maps F into ϕ_F, defined as follows:

- $\phi_F = \forall z\,(var(z) \wedge share(z,x) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_n))$ if $F = y_1 \wedge \ldots \wedge y_n \rightarrow x$.
- $\phi_F = \phi_{F_1} \wedge \ldots \wedge \phi_{F_k}$ if $F = F_1 \wedge \ldots \wedge F_k$, $k \geq 0$, and all the F_i's are definite clauses.

□

Observe that, for $n = 0$ we obtain the assertion $\forall z\,(var(z) \wedge share(z,x) \Rightarrow false)$, that is equivalent to $ground(x)$.

Example 6. The element $x \wedge (y \leftrightarrow w)$ is mapped by ε_{Def} into the assertion $ground(x) \wedge \forall z\,(var(z) \wedge share(z,w) \Rightarrow share(z,y)) \wedge \forall z\,(var(z) \wedge share(z,y) \Rightarrow share(z,w))$. □

Next, the transformation of Definition 10 is shown to be correct.

Lemma 11. $\varepsilon_{\mathrm{Def}}$ *is an embedding of* Def *into* $\mathcal{L}$.

Finally, using the above Lemma we can prove that *Def* is a representation of $\mathcal{A}_{Def}$.

Theorem 12. $\mathrm{Def} \simeq \mathcal{A}_{\mathrm{Def}}$.

In order to analyze *Def* and to compare it with other abstract domains, a maximal factorization of $\mathcal{A}_{Def}$ is given. To this end, we use the following domains. For every $|V| \geq n \geq 0$, consider the domain $\mathcal{A}_{Def^n}$ consisting of the conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z,x) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_n))$, with $y_1, \ldots, y_n$ distinct variables of V. The following result holds.

Lemma 13. $\{\mathcal{A}_{\mathrm{Def}^n} \mid n \in [0,|V|]\}$ *is a maximal factorization of* $\mathcal{A}_{\mathrm{Def}}$.

Let $\mathcal{A}_{Def^+} = \wedge_{n \in [1,|V|]} \mathcal{A}_{Def^n}$. A representation of $\mathcal{A}_{Def^+}$ is provided by the set Def^+ of positive boolean functions that can be represented as conjunctions of clauses $y_1 \wedge \ldots \wedge y_n \rightarrow x$, with $n \geq 1$, plus the bottom element *false*, with concretization function the one of *Def*. Then by Lemma 13 it follows that *Def* is (isomorphic to) the reduced-product of the domain *Con* and Def^+.

It has been recently shown in [4] that *Def* characterizes the ground-dependency information on V described by the domain *Sharing*. We shall see that this result is easily derived from the logical descriptions of these domains.

4.2 *Pos* in Logical Form

In order to study ground-dependency analysis, the abstract domain *Pos* was introduced by Marriott and Søndergaard [14, 15], consisting of the positive boolean functions, plus the bottom element *false*, with concretization function equal to γ_{Def}.

Consider the following abstract domain $\mathcal{A}_{Pos}$.

Definition 14. $\mathcal{A}_{Pos}$ is the set of assertions that are conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z,x_1) \Rightarrow Q(z,y_1,\ldots,y_n)) \vee \ldots \vee \forall z\,(var(z) \wedge share(z,x_m) \Rightarrow Q(z,y_1,\ldots,y_n))$, with $m \geq 1$, and $n \geq 0$, where $x_1,\ldots,x_m,y_1,\ldots,y_n$ are in V, and z is a fresh variable. □

We show that *Pos* is a representation of $\mathcal{A}_{Pos}$, and provide a maximal factorization (on $\mathcal{L}$) of $\mathcal{A}_{Pos}$.

First, *Pos* is characterized in logical form by means of the following transformation. We use the representation of an element F of *Pos* as a conjunction of clauses, of the form $y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$, $m \geq 1$, $n \geq 0$ (cf. [2]). In the sequel $Q(z,y_1,\ldots,y_n)$ denotes the assertion $share(z,y_1) \vee \ldots \vee share(z,y_n)$.

Definition 15. The transformation $\epsilon_{Pos} : Pos \rightarrow \mathcal{L}$ maps F into ϕ_F, defined as follows:

- $\phi_F = \forall z\,(var(z) \wedge share(z,x_1) \Rightarrow Q(z,y_1,\ldots,y_n)) \vee \ldots \vee \forall z\,(var(z) \wedge share(z,x_m) \Rightarrow Q(z,y_1,\ldots,y_n))$ if $F = y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$.
- $\phi_F = \phi_{F_1} \wedge \ldots \wedge \phi_{F_k}$ if $F = F_1 \wedge \ldots \wedge F_k$, $k \geq 0$, and all the F_i's are clauses. □

It is easy to check that the above transformation restricted to the elements of *Def* coincides with ϵ_{Def}.

Example 7. The element $x \vee y$ is mapped by ϵ_{Pos} into the assertion $\forall z\,(var(z) \wedge share(z,x) \Rightarrow false) \vee \forall z\,(var(z) \wedge share(z,y) \Rightarrow false)$, equivalent to $ground(x) \vee ground(y)$. □

Next, the transformation of Definition 15 is shown to be correct.

Lemma 16. ϵ_{Pos} *is an embedding of* Pos *into* $\mathcal{L}$.

Finally, uising Lemma 16, we can prove that *Pos* is a representation of $\mathcal{A}_{Pos}$.

Theorem 17. $\mathrm{Pos} \simeq \mathcal{A}_{\mathrm{Pos}}$.

In order to give a maximal factorization of $\mathcal{A}_{Pos}$, we use the decomposition of $\mathcal{A}_{Def}$, and the following domains. For every $|V| \geq n \geq 0$ and $|V| \geq m \geq 2$, consider the domain $\mathcal{A}_{Pos^{m,n}}$ consisting of the conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z,x_1) \Rightarrow Q(z,y_1,\ldots,y_n)) \vee \ldots \vee \forall z\,(var(z) \wedge share(z,x_m) \Rightarrow Q(z,y_1,\ldots,y_n))$ with $x_1,\ldots,x_m$ and $y_1,\ldots,y_n$ distinct variables of V. The following result holds.

Lemma 18. $\{\mathcal{A}_{\mathrm{Def}^n}, \mathcal{A}_{\mathrm{Pos}^{m,n}} \mid n \in [0,|V|], m \in [2,|V|]\}$ *is a maximal factorization of* $\mathcal{A}_{\mathrm{Pos}}$.

Let $\mathcal{A}_{Pos^\vee} = \wedge_{n\in[0,|V|],m\in[2,|V|]}\mathcal{A}_{Pos^{m,n}}$. A representation of $\mathcal{A}_{Pos^\vee}$ is provided by the set $Pos^\vee$ of positive boolean functions that can be represented as conjunctions of clauses $y_1 \wedge \ldots \wedge y_n \rightarrow x_1 \vee \ldots \vee x_m$, with $n \geq 0, m \geq 2$, plus the bottom element *false*, with concretization function the one of *Pos*. Then by Lemma 18 it follows that *Pos* is (isomorphic to) the reduced-product of the domains *Con*, Def^+ and $Pos^\vee$. It has been shown in [6] that *Def* is properly contained in *Pos*. Lemma 18 characterizes logically the other part of *Pos*.

4.3 *Sharing* in Logical Form

In order to study information on the possible sharing among abstract variables, an abstract domain extensively used in abstract interpretation is the domain *Sharing* by Jacobs and Langen [10]. *Sharing* is the set of sets $\Delta \in 2^{2^V}$ s.t. if $\Delta \neq \emptyset$ then $\emptyset \in \Delta$. Its concretization function $\gamma_{Sharing}$ maps an element Δ of *Sharing* into the set $\gamma_{Sharing}(\Delta)$ of those substitutions σ whose approximation set $A(\sigma)$ is an element of Δ. The approximation set $A(\sigma)$ consists of all the sets $occ(\sigma, x) = \{v \mid v \text{ in the domain of } \sigma \text{ s.t. } x \text{ occurs in } v\sigma\}$, for all the variables x occurring in the range of σ.

Consider the following abstract domain $\mathcal{A}_{Sharing}$.

Definition 19. $\mathcal{A}_{Sharing}$ is the set of assertions of $\mathcal{L}$ that are conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z, x_1) \wedge \ldots \wedge share(z, x_m) \Rightarrow share(z, y_1) \vee \ldots \vee share(z, y_n))$ with $m \geq 1$, $n \geq 0$, where $x_1, \ldots, x_m, y_1, \ldots, y_k$ are in V, and z is a fresh variable. □

We show that *Sharing* is a representation of $\mathcal{A}_{Sharing}$, and provide a maximal factorization (on $\mathcal{L}$) of $\mathcal{A}_{Sharing}$.

First, *Sharing* is characterized in logical form by means of the following transformation. In the sequel, for the sake of simplicity, we write $share(x, S)$ instead of $share(\{x\} \cup S)$.

Definition 20. The transformation $\varepsilon_{Sharing}$ maps Δ into the assertion

$$\phi_\Delta = \bigwedge_{S \subseteq V} \forall z\,(var(z) \wedge share(z, S) \Rightarrow share(z, S_1) \vee \ldots \vee share(z, S_k)),$$

with $\{S_1, \ldots, S_k\} = \{S' \mid S' \in \Delta \text{ s.t. } S \subseteq S'\}$.

□

Let ϕ_S denote the conjunct of ϕ_Δ corresponding to the subset S of V.

Observe that if S is not contained in any set of Δ, then ϕ_S is the assertion $\forall z\,(var(z) \wedge share(z, S) \Rightarrow false)$, which says that the variables of S can only be bound to terms sharing no variables. If S is a singleton, say $S = \{x\}$, then ϕ_S describes information on ground-dependency for x. Indeed, it is not difficult to see that in this case ϕ_S can be rewritten into an assertion of $\mathcal{A}_{Def}$. The other assertions ϕ_S, for S not singleton and $k > 0$, describe information about sharing of sets containing at least three variables.

Example 8. Consider $\Delta = \{\emptyset, \{x\}, \{x, y\}, \{y, z\}\}$, and $V = \{x, y, z\}$. Then ϕ_Δ is (equivalent to) $\neg share(x, z) \wedge \neg share(\{x, y, z\}) \wedge \forall v(var(v) \wedge share(v, y) \Rightarrow share(v, z) \vee share(v, x)) \wedge \forall v(var(v) \wedge share(v, z) \Rightarrow share(v, y))$. □

Next, the correctness of this transformation is shown.

Lemma 21. $\varepsilon_{\text{Sharing}}$ *is an embedding of* Sharing *into* $\mathcal{L}$.

Finally, Lemma 21 is used to prove that *Sharing* is a representation of $\mathcal{A}_{Sharing}$. In the proof, we use the fact that the assertion $\forall z\,(var(z) \wedge share(z,S) \Rightarrow share(z,S_1) \vee \ldots \vee share(z,S_k))$ is equivalent to the assertion consisting of the conjunction of the formulas $\forall z\,(var(z) \wedge share(z,x_1) \wedge \ldots \wedge share(z,x_m) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_k))$, for all $(y_1,\ldots,y_k)$ occurring in $S_1 \times \ldots \times S_k$.

Theorem 22. Sharing $\simeq \mathcal{A}_{\text{Sharing}}$.

In order to give a maximal factorization of $\mathcal{A}_{Sharing}$, we use the following domains. For every $|V| \geq n \geq 0$ and $|V| \geq m \geq 1$, consider the domain $\mathcal{A}_{Sharing^{m,n}}$ consisting of the conjunctions of formulas of the form $\forall z\,(var(z) \wedge share(z,x_1) \wedge \ldots \wedge share(z,x_m) \Rightarrow share(z,y_1) \vee \ldots \vee share(z,y_n))$, with $x_1,\ldots,x_m$ and $y_1,\ldots,y_n$ distinct variables of V. The following result holds.

Lemma 23. $\{\mathcal{A}_{\text{Sharing}^{m,n}} \mid n \in [0,|V|], m \in [1,|V|]\}$ *is a maximal factorization of* $\mathcal{A}_{\text{Sharing}}$.

Consider the abstract domain $Sharing^+$ introduced in [4], containing as elements the empty set, and the sets Δ^+ of the form $\Delta \cup T$, with Δ in *Sharing* and $T = \{\{x\} \mid x \in V\} \cup \{\emptyset\}$. One can prove that $Sharing^+$ is a representation of $\bigwedge_{m\geq 2, n\geq 0} \mathcal{A}_{Sharing^{m,n}}$. Moreover, *Def* is a representation of $\bigwedge_{n\geq 0} \mathcal{A}_{Sharing^{1,n}}$. Therefore, by Lemma 23 it follows that *Sharing* is (isomorphic to) the reduced product of $Sharing^+$,Def^+ and *Con*.

4.4 *ASub* in Logical Form

The pair-sharing domain *ASub* was introduced by Søndergaard [18] for sharing and linearity analysis. Its elements are pairs (G,R) where the first component is a subset of V, and the second one is a symmetric binary relation on V, s.t. $(G \times V) \cap R = \emptyset$. Moreover, the element $\bot$, representing the empty set of substitutions, is in *ASub*. Its concretization function γ_{ASub} maps an element (G,R) of *ASub* into the set of substitutions σ s.t. for all (x,y) in V: (i) x in G implies $x\sigma$ ground; (ii) x,y distinct and $OVar(x\sigma) \cap OVar(y\sigma) \neq \emptyset$ implies (x,y) in R; (iii) $(x,x) \notin R$ implies $x\sigma$ linear.

Consider the following abstract domain $\mathcal{A}_{ASub}$.

Definition 24. $\mathcal{A}_{ASub}$ is the set of assertions that are conjunctions of literals of the form $ground(x)$, $\neg share(x,y)$, and $linear(x)$, with x,y in V. □

We show that *ASub* is a representation of $\mathcal{A}_{ASub}$, and provide a maximal factorization of $\mathcal{A}_{ASub}$.

First, *ASub* is characterized in logical form by means of the following transformation.

Definition 25. The transformation ε_{ASub} maps $\bot$ into *false*, and (G,R) into the assertion $\phi_{(G,R)} = \phi_1 \wedge \phi_2 \wedge \phi_3$, where:

1. ϕ_1 is the conjunction of the atoms $ground(x)$, for all x in G.
2. ϕ_2 is the conjunction of the literals $\neg share(x, y)$, for all (x, y) not in R with x, y distinct.
3. ϕ_3 is the conjunction of the atoms $linear(x)$, for all (x, x) not in R. □

Assertions ϕ_1, ϕ_2 and ϕ_3 characterize $ASub$ in logical form, by means of its information on groundness, independence, and linearity, respectively.

Example 9. Consider the element (G, R) of $ASub$, with $G = \{x\}$ and $R = \{(y, z), (z, z), (z, w)\}$ and suppose that $V = \{x, y, z, w\}$. Then $\phi_{(G,R)}$ is (equivalent to) $ground(x) \wedge linear(y) \wedge linear(w) \wedge \neg share(y, w)$. □

Next, this transformation is shown to be correct.

Lemma 26. $\varepsilon_{\mathrm{ASub}}$ *is an embedding of* ASub *into* $\mathcal{L}$.

Finally, Lemma 26 is used to prove that $ASub$ is a representation of $\mathcal{A}_{ASub}$.

Theorem 27. $\mathrm{ASub} \simeq \mathcal{A}_{\mathrm{ASub}}$.

In order to give a maximal factorization of $\mathcal{A}_{ASub}$, the domain $\mathcal{A}_{Linear}$ is used, consisting of the conjunctions of atoms the form $linear(x)$, with x in V.

Lemma 28. $\{\mathcal{A}_{\mathrm{Sharing}^{m,0}}, \mathcal{A}_{\mathrm{Linear}} \mid m \in [1, 2]\}$ *is a maximal factorization of* $\mathcal{A}_{\mathrm{ASub}}$.

5 Conclusion

In this paper a simple framework based on first-order logic has been proposed for reasoning about abstract domains for static analysis. The notions of domain representation and of conjunctive factorization have been introduced for analyzing and comparing abstract domains. The usefulness of this framework has been illustrated by considering a number of abstract domains used in abstract interpretation of logic programs.

We discuss now some related work.

In [7], the Galois insertion approach is used to define the notion of quotient of a domain D w.r.t. another domain P, describing the part of D that is useful for computing the information described by P. In this paper the logical characterization and factorization of the domains allows to perform a similar analysis, where D and P are first characterized logically, and next factorized. Then the reduced product of the common factors of the domains corresponds to the quotient of D w.r.t. P.

In [4] the approach based on closure operators is used. To this end, the lattice of all the abstract domains (according with the original definition of the Cousots, cf. [3]) is considered. Abstract domains are decomposed by means of the notion of pseudo-complement, a kind of inverse of the reduced-product. Instead, in our method the set of abstract domains considered depends on the set V of the

program variables, as well as on the class of properties described in the assertion language. Moreover, we use a direct approach for decomposing a domain, by inspecting the syntactic form of the relative set of assertions.

The abstract domains analysed in Section 4 have been extensively studied in previous work. In [6] it is proven that the part of *Sharing* describing groundness dependencies is contained in *Pos*. In [4] this result is strengthen by showing that this part coincides with *Def*, and that $Sharing^+$ is the pseudo-complement of *Def* in *Sharing*. In this paper these results are directly derived from the logical characterization of *Sharing*. Moreover, we have obtained the finest (in L) decomposition of *Sharing*. Finally, the factors of this decomposition have been used for other purposes, e.g. for comparing *Sharing* with *ASub*.

The classes of Boolean functions used to represent *Def* and *Pos* have been extensively analyzed (e.g. [5, 2]). The difference from these works is that they focus on the representation, while we focus on the design and reasoning, by considering a syntactic characterization in first-order logic of their image under the concretization function.

An interesting topic that seems worth of investigation, is the study of the relationship between abstract interpretations and proof methods. This topic has been tackled in the functional programming setting, where a domain-theoretic approach is used in [11] for proving that strictness analysis by abstract interpretation and non-standard type inference are equivalent. For logic programming, our framework could be used for defining a program logic for the comparison of data-driveness analysis using type inference (cf. e.g. [1]) and abstract interpretation (cf. [16]).

Acknowledgements

This work was partially supported by NWO, the Dutch Organization for Scientific Research, under grant 612-32-001. I would like to thank the referees for their useful comments; Livio Colussi, Tino Cortesi, Gilberto Filé, Maurizio Gabbrielli, Massimo Marchiori, and Catuscia Palamidessi for stimulating discussions on the subject of this paper; and Jan Rutten for his encouragement.

References

1. K.R. Apt and E. Marchiori. Reasoning about Prolog programs: from Modes through types to assertions. In Formal Aspects of Computing, vol. 6A, pag. 743-764, 1994.
2. T. Armstrong, K. Marriott, P. Schachte and H. Søndergaard. Boolean Functions for Dependency Analysis: Algebraic Properties and Efficient Representation. Proc. *SAS'94*, B. Le Charlier ed., Springer-Verlag, LNCS 864, pp. 266-280, 1994.
3. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. Proc. *POPL '79*, pp. 269–282. ACM Press, 1979.
4. A. Cortesi, G. Filé, R. Giacobazzi, C. Palamidessi, and F. Ranzato. Complementation in abstract interpretation. Proc. *SAS '95*, A. Mycroft ed., LNCS Vol. 983, pp. 100–117. Springer-Verlag, 1995.

5. A. Cortesi, G. Filé, and W. Winsborough. Prop revised: propositional formula as abstract domain for groundness analysis. Proc. *LICS '91*, G. Kahn ed., pp. 322–327, 1991.
6. A. Cortesi, G. Filé, and W. Winsborough. Comparison of abstract interpretations. Proc. *ICALP '92*, W. Kuich, ed. LNCS Vol. 623, pp. 521–532. Springer-Verlag, 1992.
7. A. Cortesi, G. Filé, and W. Winsborough. The quotient of an abstract interpretation. Technical Report 12/94, Dipartimento di Matematica Pura ed Applicata, Università di Padova, 1994.
8. P. Dart. On derived dependencies and connected databases. *Journal of Logic Programming*, 11(2):163–188, 1991.
9. S. Debray. On the Complexity of Dataflow Analysis of Logic Programs. Proc. *ICALP '92*, Springer Verlag, pp. 509-520, LNCS 623, 1992.
10. D. Jacobs and A. Langen. Static analysis of logic programs for independent AND-parallelism. *Journal of Logic Programming*, 13(2,3):154–165, 1992.
11. T.P. Jensen. Strictness Analysis in Logical Form. Proc. *Conference on Functional Programming Languages and Computer Architectures*, Springer Verlag, pp. 352-366, LNCS 523, 1991.
12. N.D. Jones and H. Søndergaard. A Semantics-Based Framework for the Abstract Interpretation of Prolog. *Abstract Interpretation of Declarative Languages*, eds. S. Abramsky and C. Hankin, Ellis Horwood, Chichester, U.K., pp. 123-142, 1987.
13. E. Marchiori. A Logic for Variable Aliasing in Logic Programs. Proc. *ALP '94*, G. Levi, M. Rodriguez-Artalejo eds., Springer Verlag, pp. 287-304, LNCS 850, 1994.
14. K. Marriott and H. Søndergaard. Notes for a tutorial on abstract interpretation of logic programs. *North American Conference on Logic Programming*, 1989.
15. K. Marriott and H. Søndergaard. Precise and efficient groundness analysis for logic programs. *ACM LoPLaS*, 2(1–4):181–196, 1993.
16. K. Marriott, H. Søndergaard and N.D. Jones. Denotational abstract interpretation of logic programs. *ACM Transactions on Programming Languages and Systems*, ACM-TOPLAS, 16(3):607–648, 1994.
17. C. Mellish. Some Global Optimizations for a Prolog Compiler. *The Journal of Logic Programming*, 2(1):43–66, 1985.
18. H. Søndergaard. An Application of Abstract Interpretation of Logic Programs: Occur Check Reduction. Proc. *ESOP '86*, eds. B. Robinet and R. Wilhelm, Springer Verlag, pp. 327-338, LNCS 213, 1986.